ARMY RESEARCH LABORATORY

A One-Dimensional, Noniterative Trajectory Model (With a C++ Implementation)

by Robert J. Yager and Benjamin J. Flanders

ARL-TN-598 **March 2014**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-598 **March 2014**

A One-Dimensional, Noniterative Trajectory Model (With a C++ Implementation)

Robert J. Yager and Benjamin J. Flanders
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2014	Final	July 2012–June 2013

4. TITLE AND SUBTITLE
A One-Dimensional, Noniterative Trajectory Model (With a C++ Implementation)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Robert J. Yager and Benjamin J. Flanders

5d. PROJECT NUMBER
AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-WML-A
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-598

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report defines algorithms that can be used to calculate speed, distance, and time variables as functions of initial and/or final projectile states. Examples include speed as a function of initial speed and distance traveled, distance traveled as a function of initial and final speeds, and time of flight as a function of initial and final speeds. Derivations for the algorithms are based exclusively on force due to air resistance and allow for drag coefficients that have a functional relationship to Mach number.

15. SUBJECT TERMS
fragment, trajectory, C++, gravity, analytic, noniterative, time of flight, Mach

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert J. Yager
Unclassified	Unclassified	Unclassified	UU	30	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6689

Contents

List of Figures

List of Tables

Acknowledgments

The authors would like to thank Dr. Paul Weinacht of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Dr. Weinacht provided technical and editorial recommendations that improved the quality of this report.

1. Introduction

This report defines algorithms that can be used to calculate speed, distance, and time variables as functions of initial and/or final projectile states. Examples include speed as a function of initial speed and distance traveled, distance traveled as a function of initial and final speeds, and time of flight as a function of initial and final speeds. Derivations for the algorithms are based exclusively on force due to air resistance and allow for drag coefficients that have a functional relationship to Mach number.

All algorithms are written in terms of scaled variables (speed, distance, and time) that are independent of units. This approach allows for the precalculation of a set of parameters that depend only on the characteristics of a particular drag versus Mach curve (i.e., the parameters are independent of projectile mass, projectile cross-sectional area, and air density). A practical benefit of this approach is a reduction in computation time.

The algorithms presented in this report are particularly useful for situations where total distance traveled is approximately equal to the distance between starting and ending locations. Examples include flat-fire trajectories and fragment trajectories.

JTCG/ME-79-1-2[1] presents a method for calculating speed as a function of initial speed and total distance traveled. It relies upon an analytic solution that is similar to what is presented in sections 3.1 and 3.2 of this report. The key difference is that the method presented in this report provides a solution to the equations of motion in terms of variables that are independent of units.

2. Scaled Variables

Recall that Mach number, M, is defined to be the ratio of the speed, v, of an object moving through a fluid to the local speed of sound, v_s. That is,

$$M \equiv \frac{v}{v_s} \tag{1}$$

Scaled distance, s, is defined as a function of measured distance, x:

[1]Joint Technical Coordinating Group for Munitions Effectiveness. *Computer Program for General Full Spray Materiel MAE Computations, Volume II – Analyst Manual;* AMSAA 61 JTCG/ME-79-1-2; U.S. Army Materiel Systems Analysis Activity: Aberdeen, MD, 1987.

$$s \equiv \frac{\rho A}{2m} x \tag{2}$$

where ρ is the density of the air through which a projectile travels, A is the projectile's cross-sectional area, and m is its mass.

Scaled time, τ, is defined as a function of measured time, t:

$$\tau \equiv \frac{\rho A v_s}{2m} t \tag{3}$$

3. Derivation of Equations for Linear Drag Coefficients

Begin with the equation for the force acting on a projectile that is due to air resistance:

$$F = -\frac{1}{2} C_D \rho A v^2 \tag{4}$$

where C_D is a projectile's drag coefficient, and ρ, A, and v are as defined in section 2.

3.1 Drag Coefficient as a Function of Mach Number

Assume that C_D is linearly dependent on Mach number (figure 1). Then

$$C_D = \alpha M + \beta \tag{5}$$

where α and β are constants representing slope and intercept, respectively.

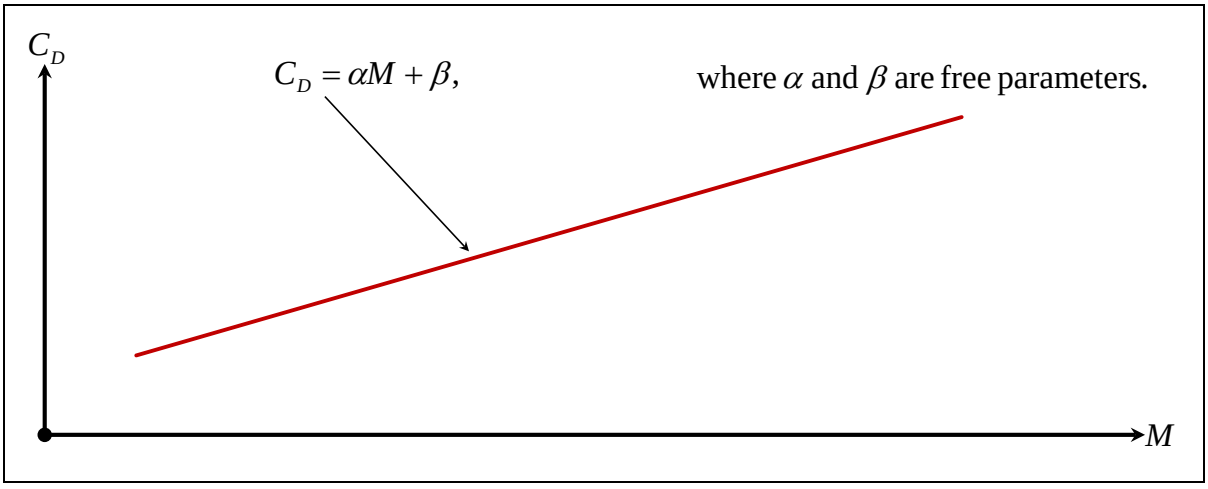


Figure 1. Drag coefficient, C_D, as a function of Mach number, M.

3.2 Scaled Distance as a Function of Mach Number

Using Newton's second law, equation 4 can be used to solve for acceleration:

$$F = ma \tag{6}$$

$$\Rightarrow a = -\frac{1}{2m} C_D \rho A v^2 \tag{7}$$

Next, the definitions of speed and acceleration can be used to find acceleration as a function of speed and position:

$$v \equiv \frac{dx}{dt} \tag{8}$$

$$\text{and } a \equiv \frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} \tag{9}$$

$$\Rightarrow a = v\frac{dv}{dx} \tag{10}$$

Combining equations 7 and 10,

$$\frac{dv}{dx} = -\frac{1}{2m} C_D \rho A v \tag{11}$$

Next, equations 1 and 2 can be used to rewrite equation 11 in terms of scaled variables:

$$\frac{dM}{ds} = -C_D M \tag{12}$$

Substituting equation 5 into equation 12,

$$\frac{dM}{ds} = -(\alpha M + \beta)M \tag{13}$$

Equation 13 can be solved for *s* by using the method of separation of variables:

$$s = -\int \frac{1}{(\alpha M + \beta)M}\, dM \tag{14}$$

The solution to the integral in equation 14 can be found in integral tables:[2]

$$s = \frac{1}{\beta}\ln\left(\alpha + \frac{\beta}{M}\right) + c \tag{15}$$

where *c* is a constant of the integration and is determined by initial values for *s* and *M*.

[2]Lide, D. R. *Handbook of Chemistry and Physics*, 89th ed.; CRC Press: London, 2008; p A-16, equation 37.

$$c = s_0 - \frac{1}{\beta}\ln\left(\alpha + \frac{\beta}{M_0}\right) \tag{16}$$

Note that equations 15 and 16 are only valid for $\alpha + \beta/M > 0$, $M \neq 0$, and $\beta \neq 0$. Since, by definition, Mach numbers are non-negative, the conditions $\alpha + \beta/M > 0$ and $M \neq 0$ are assured if we assume that $C_D > 0$ for all values of M and $M_0 > 0$. The case where $\beta = 0$ will occur occasionally. To be exact, equation 14 should be solved for the special case where $\beta = 0$. In practice, however, when encountering a situation where $\beta = 0$, it is sufficient to change β to some very small value.

Figure 2 presents scaled distance, *s*, as a function of Mach number, *M*.

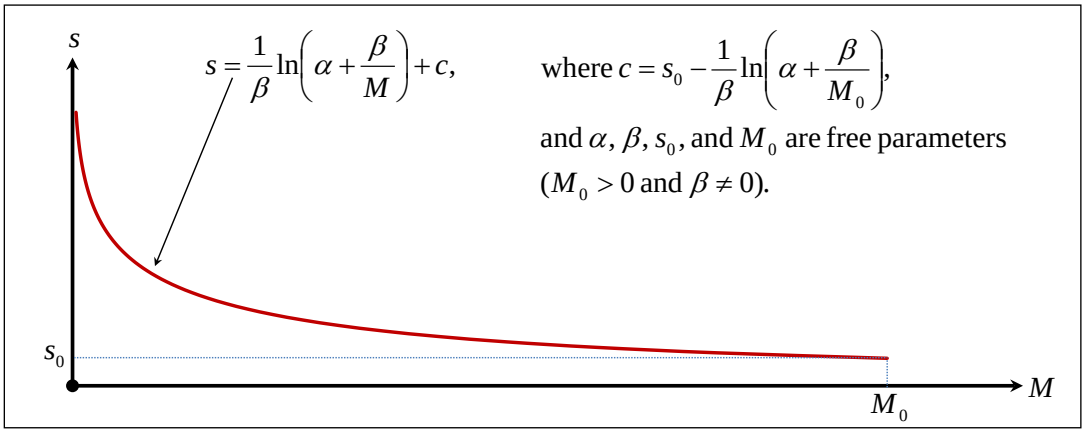


Figure 2. Scaled distance, *s*, as a function of Mach number, *M*.

3.3 Mach Number as a Function of Scaled Distance

Solving equation 15 for *M* yields Mach number as a function of scaled distance (figure 3):

$$M = \frac{\beta}{e^{\beta(s-c)} - \alpha} \tag{17}$$

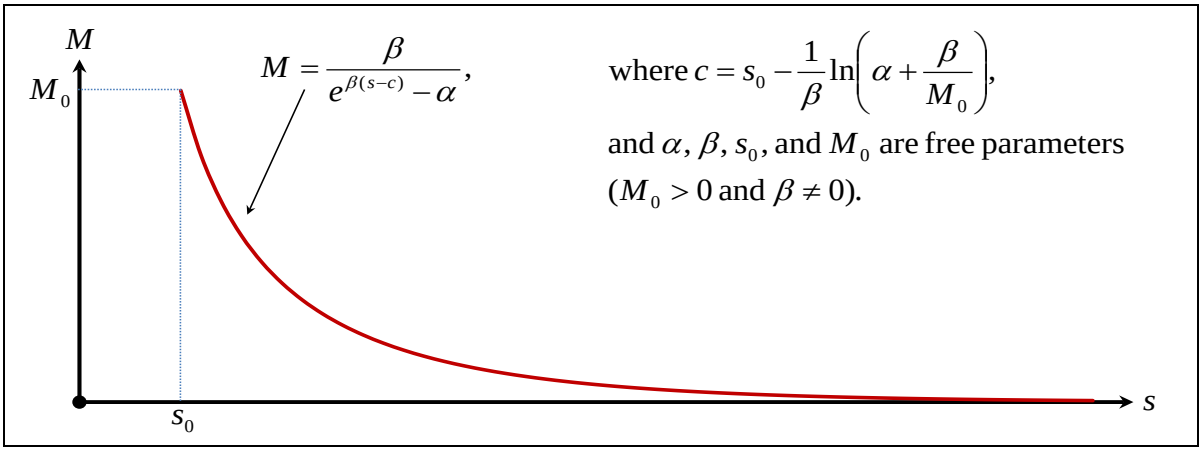


Figure 3. Mach number, *M*, as a function of scaled distance, *s*.

3.4 Scaled Time as a Function of Mach Number

Begin by using equations 1 and 7 to find acceleration as a function of Mach number:

$$a = -\frac{1}{2m} C_D \rho A v_s^2 M^2 \tag{18}$$

Next, use equations 1 and 9 to relate acceleration to the time derivative of Mach number:

$$a = v_s \frac{dM}{dt} \tag{19}$$

Combining equations 18 and 19,

$$\frac{dM}{dt} = -\frac{1}{2m} C_D \rho A v_s M^2 \tag{20}$$

Equation 3 can be used to rewrite equation 20 in terms of scaled time:

$$\frac{dM}{d\tau} = -C_D M^2 \tag{21}$$

Substituting equation 5 into equation 21,

$$\frac{dM}{d\tau} = -(\alpha M + \beta) M^2 \tag{22}$$

Equation 22 can be solved for τ by using the method of separation of variables:

$$\tau = -\int \frac{1}{(\alpha M + \beta) M^2} dM \tag{23}$$

The solution to the integral in equation 23 can be found in integral tables:[3]

$$\tau = \frac{1}{\beta M} - \frac{\alpha}{\beta^2} \ln\left(\alpha + \frac{\beta}{M}\right) + d \tag{24}$$

where d is a constant of the integration and is determined by initial values for τ and M.

$$d = \tau_0 - \frac{1}{\beta M_0} + \frac{\alpha}{\beta^2} \ln\left(\alpha + \frac{\beta}{M_0}\right) \tag{25}$$

As was the case with equations 15 and 16, equations 24 and 25 are only valid for $\alpha + \beta / M > 0$, $M \neq 0$, and $\beta \neq 0$. Since, by definition, Mach numbers are non-negative, the conditions $\alpha + \beta / M > 0$ and $M \neq 0$ are assured if we assume that $C_D > 0$ for all values of M and $M_0 > 0$.

[3] Lide, D. R. *Handbook of Chemistry and Physics*, 89th ed.; CRC Press: London, 2008; pp A-17, equation 40.

The case where $\beta = 0$ will occur occasionally. To be exact, equation 23 should be solved for the special case where $\beta = 0$. In practice, however, when encountering a situation where $\beta = 0$, it is sufficient to change β to some very small value.

Figure 4 presents scaled time, τ, as a function of Mach number, *M*.

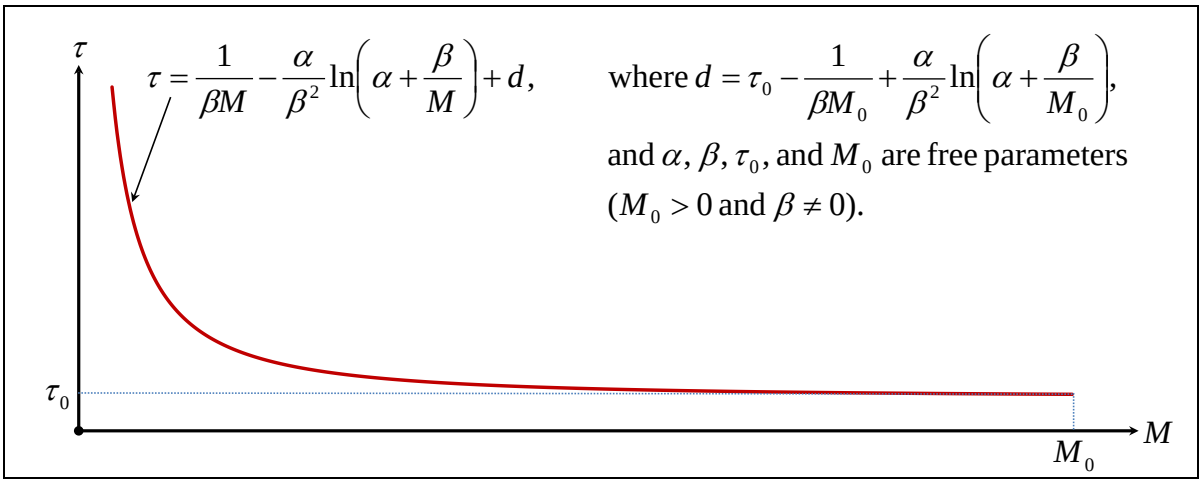


Figure 4. Scaled time, τ, as a function of Mach number, *M*.

4. Generalization to an Arbitrary Drag-Mach Relationship

The equations in section 3 can be generalized for a piecewise, linear function of drag given Mach. This technique allows for any functional relationship between drag coefficient and Mach number to be modeled.

4.1 Drag Coefficient as a Piecewise, Linear Function of Mach Number

Suppose that drag coefficient as a function of Mach number is defined by *n* matched pairs of values for Mach number and drag coefficient, $(M_k, C_{D,k})$, such that

$$M_{k+1} < M_k \text{ for } 0 \le k < n-1 \tag{26}$$

Using the matched pairs, $n-1$ equations that have the form of equation 5 can be generated:

$$C_D = \alpha_k M + \beta_k \tag{27}$$

for k such that $M_{k+1} \le M \le M_k$ and $0 \le k < n-1$.

Figure 5 presents drag coefficient, C_D, as a piecewise function of Mach number, *M*.

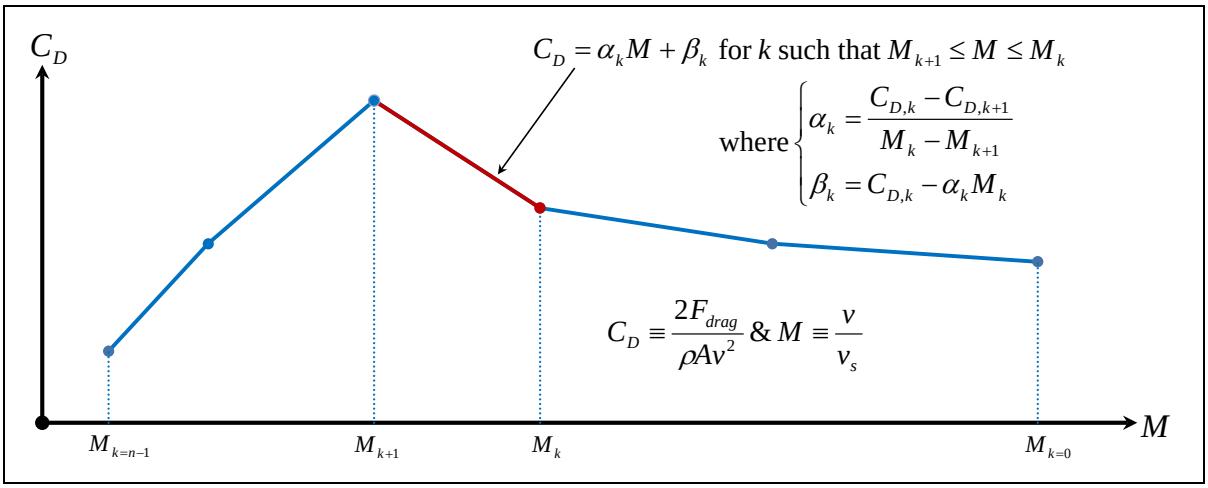


Figure 5. Drag coefficient, C_D, as a piecewise function of Mach number, M.

Equation 27 can be used to determine the values of α_k and β_k:

$$\alpha_k = \frac{C_{D,k} - C_{D,k+1}}{M_k - M_{k+1}} \tag{28}$$

and

$$\beta_k = C_{D,k} - \alpha_k M_k \tag{29}$$

4.2 Scaled Distance as a Function of Mach Number

Equation 15 can be generalized to a piecewise function of Mach number:

$$s = \frac{1}{\beta_k} \ln\left(\alpha_k + \frac{\beta_k}{M}\right) + c_k \tag{30}$$

for k such that $M_{k+1} \le M \le M_k$.

Values for c_k can be found by generalizing equation 16:

$$c_k = s_{0,k} - \frac{1}{\beta_k} \ln\left(\alpha_k + \frac{\beta_k}{M_{0,k}}\right) \tag{31}$$

Values for initial conditions for each segment of the function are given by equations 32 and 33:

$$M_{0,k} = M_k \tag{32}$$

and

$$s_{0,k} = s_k \tag{33}$$

Values for s_k can be obtained from equation 30:

$$s_k = \frac{1}{\beta_{k-1}} \ln\left(\alpha_{k-1} + \frac{\beta_{k-1}}{M_k}\right) + c_{k-1} \tag{34}$$

Since the value of $s_{k=0}$ is arbitrary, zero is used as a convenient choice:

$$s_{k=0} = 0 \tag{35}$$

Figure 6 presents an overview of the functions necessary to calculate scaled distance as a function of Mach number. Note that for a particular drag coefficient versus Mach table, α_k, β_k, c_k, and s_k are unique (independent of particle mass, air density, etc.). Thus, they can be precalculated and stored in tabulated form.

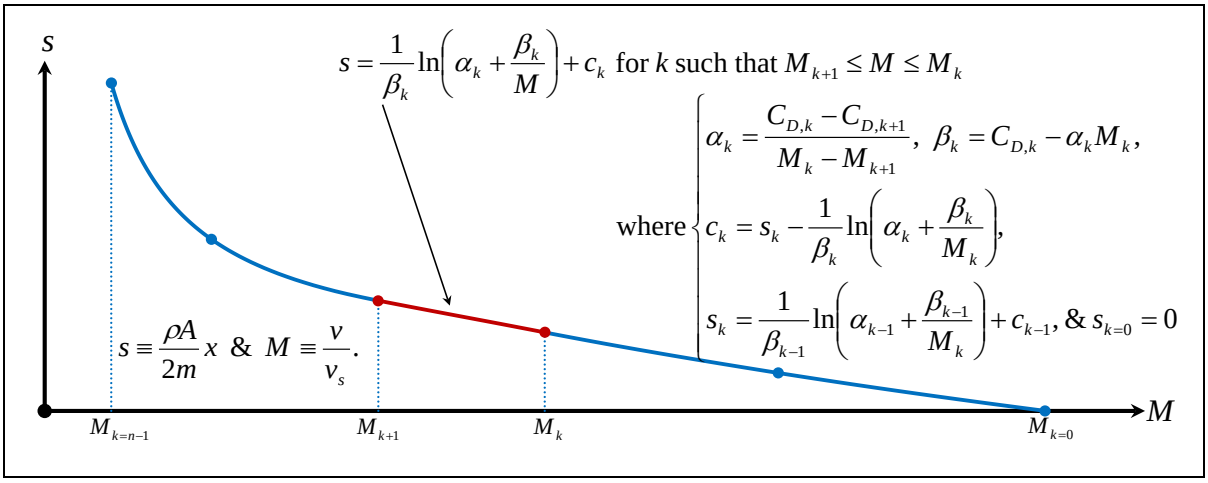


Figure 6. Scaled distance, *s*, as a piecewise function of Mach number, *M*.

4.3 Mach Number as a Function of Scaled Distance

Equation 17 can be generalized to a piecewise function of scaled distance:

$$M = \frac{\beta_k}{e^{\beta_k (s - c_k)} - \alpha_k} \tag{36}$$

for k such that $s_k \le s \le s_{k+1}$.

Figure 7 presents an overview of the functions necessary to calculate Mach as a function of scaled distance. Note that for a particular drag coefficient versus Mach table, α_k, β_k, c_k, and s_k are unique (independent of particle mass, air density, etc.). Thus, they can be precalculated and stored in tabulated form.

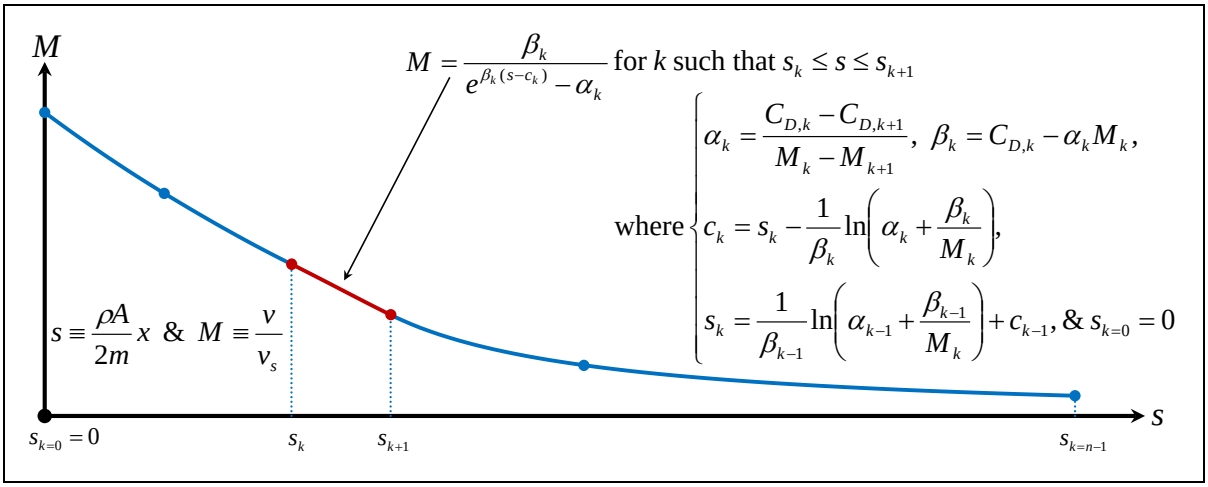


Figure 7. Mach number, *M*, as a piecewise function of scaled distance, *s*.

4.4 Scaled Time as a Function of Mach Number

Equation 24 can be generalized to a piecewise function of Mach number:

$$\tau = \frac{1}{\beta_k M} - \frac{\alpha_k}{\beta_k^2} \ln\left(\alpha_k + \frac{\beta_k}{M}\right) + d_k \tag{37}$$

for k such that $M_{k+1} \le M \le M_k$.

Values for d_k can be found by generalizing equation 25:

$$d_k = \tau_{0,k} - \frac{1}{\beta_k M_{0,k}} + \frac{\alpha_k}{\beta_k^2} \ln\left(\alpha_k + \frac{\beta_k}{M_{0,k}}\right) \tag{38}$$

Values for initial conditions for each segment of the function are given by equations 32 and 39:

$$\tau_{0,k} = \tau_k \tag{39}$$

Values for τ_k can be obtained from equation 37:

$$\tau_k = \frac{1}{\beta_{k-1} M_k} - \frac{\alpha_{k-1}}{\beta_{k-1}^2} \ln\left(\alpha_{k-1} + \frac{\beta_{k-1}}{M_k}\right) + d_{k-1} \tag{40}$$

Since the value of $\tau_{k=0}$ is arbitrary, zero is used as a convenient choice:

$$\tau_{k=0} = 0 \tag{41}$$

Figure 8 presents an overview of the functions necessary to calculate scaled time as a function of Mach number. Note that given the choice of $\tau_{k=0} = 0$, for a particular drag versus Mach table, α_k, β_k, d_k, and τ_k are unique. Thus, they can be precalculated and stored in tabulated form.

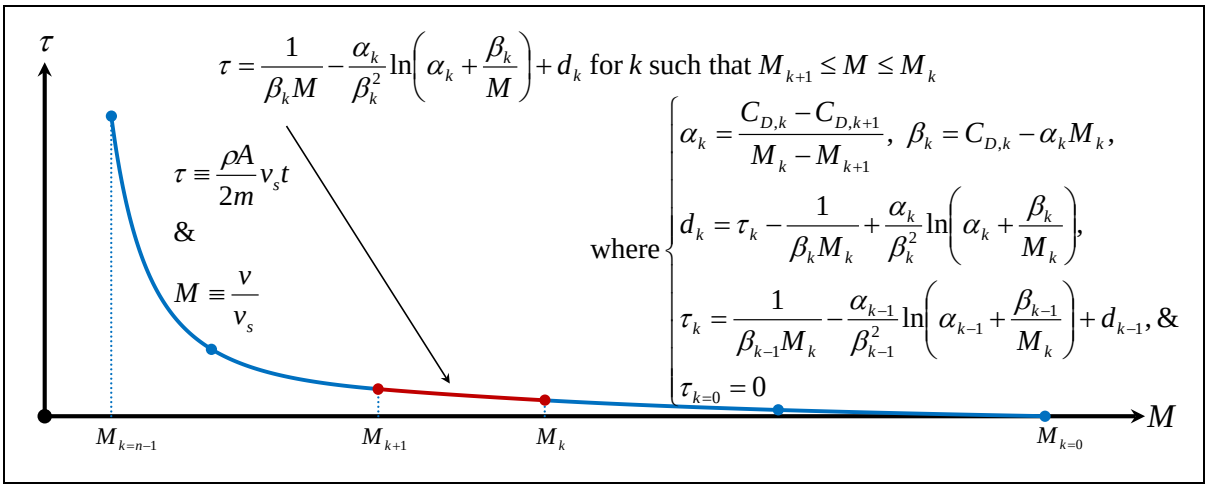


Figure 8. Scaled time, τ, as a piecewise function of Mach number, *M*.

5. State Variables and Functional Notation

For a given set of atmospheric and projectile characteristics, knowledge of any one of the three scaled variables defined in section 2 (i.e., *M*, *s*, or τ) is sufficient to uniquely define the state of a projectile. Applications of the equations presented in section 4 typically involve solving for some unknown state variable given one or two known state variables. The following six equations can make solving for unknowns easier.

5.1 State Variables

Since *M*, *s*, and τ can each be used to characterize the state of a projectile, the following three definitions can be used to quantify a change in a state variable.

$$\Delta M \equiv M_{final} - M_{initial} \tag{42}$$

$$\Delta s \equiv s_{final} - s_{initial} \tag{43}$$

$$\Delta \tau \equiv \tau_{final} - \tau_{initial} \tag{44}$$

5.2 Functional Notation

Using equation 30, define *s*(*M*) to be scaled distance as a function of Mach number:

$$s(M) = \frac{1}{\beta_k}\ln\left(\alpha_k + \frac{\beta_k}{M}\right) + c_k \tag{45}$$

Using equation 36, define *M*(*s*) to be Mach number as a function of scaled distance:

$$M(s) = \frac{\beta_k}{e^{\beta_k (s - c_k)} - \alpha_k} \tag{46}$$

Using equation 37, define $\tau(M)$ to be scaled time as a function of Mach number:

$$\tau(M) = \frac{1}{\beta_k M} - \frac{\alpha_k}{\beta_k^2} \ln\left(\alpha_k + \frac{\beta_k}{M}\right) + d_k \tag{47}$$

6. Solving for Unknowns

When attempting to solve for some unknown state variable, it helps to break the problem into four sections:

- Identify known and unknown values for unscaled state variables.
- Convert known values for state variables to scaled state variables using the equations presented in section 2.
- Use the equations presented in section 5 to develop an equation that is solved for the unknown scaled state variable in terms of known scaled state variables.
- Use the equations presented in section 2 to convert the unknown scaled state variable to an unscaled value.

For example, suppose that initial speed, $v_{initial}$, and distance traveled, Δx, are known, and it is wished to determine final speed, v_{final}. The first step is to use equations 1 and 2 to convert $v_{initial}$ and Δx to the scaled variables $M_{initial}$ and Δs. Next, represent the problem in functional notation:

$$M_{final} = M(s_{final}) \tag{48}$$

where $M(s_{final})$ is given by equation 46.

Solve equation 42 for the final scaled speed and substitute into equation 46:

$$M_{final} = M(\Delta s + s_{initial}) \tag{49}$$

Next, note that $s_{initial}$ can be calculated using equation 45. Thus,

$$M_{final} = M(\Delta s + s(M_{initial})) \tag{50}$$

Finally, use equation 1 to find v_{final} given M_{final}. A wide variety of problems can be solved in this manner.

7. Example: The Lapua Scenar GB528 Rifle Bullet

7.1 Creating a Table of Unitless Constants

Wikipedia's External Ballistics page[4] lists drag coefficient versus Mach number values for the Lapua Scenar GB528 rifle bullet. Using those values, equations from section 4 can be used to calculate the parameters listed in table 1. Specifically, given M_k and $C_{D,k}$ for $0 \le k < 28$, the equations summarized in figures 7 and 8 have been used to calculate α_k, β_k, s_k, c_k, τ_k, and d_k.

Table 1. Unitless parameters for a Lapua Scenar GB528 rifle bullet.

k	M	C_D	α	β	s	c	τ	d
0	2.4	0.27	-0.06	0.414	0	5.2773	0	-0.241615
1	2.2	0.282	-0.05	0.392	0.315209	5.555784	0.137177	-0.353937
2	2	0.292	-0.06	0.412	0.647242	5.317506	0.295484	-0.237973
3	1.8	0.304	-0.085	0.457	1.000723	4.89244	0.481805	-0.01001
4	1.6	0.321	-0.07	0.433	1.377521	5.087262	0.703845	-0.139846
5	1.5	0.328	-0.08	0.448	1.576392	4.96971	0.832223	-0.049923
6	1.4	0.336	-0.07	0.434	1.784183	5.072469	0.975621	-0.13983
7	1.3	0.343	-0.05	0.408	2.002448	5.268108	1.137427	-0.347739
8	1.2	0.348	0	0.348	2.234102	5.791212	1.32293	-1.071706
9	1.15	0.348	0.02	0.325	2.3564	6.034291	1.427044	-1.474873
10	1.1	0.347	0.08	0.259	2.48432	6.938917	1.54079	-3.345151
11	1.075	0.345	0.16	0.173	2.550765	9.120311	1.601894	-9.851053
12	1.05	0.341	0.28	0.047	2.619369	26.548368	1.666471	-161.15269
13	1.025	0.334	1.12	-0.814	2.690775	1.313248	1.735305	1.038474
14	1	0.306	2.8	-2.494	2.768002	2.293195	1.811601	1.679499
15	0.975	0.236	2.36	-2.065	2.862002	2.175026	1.906853	1.618416
16	0.95	0.177	0.92	-0.697	2.988737	0.577959	2.038621	0.366765
17	0.925	0.154	0.48	-0.29	3.150184	-3.032027	2.210906	-4.293853
18	0.9	0.142	0.2	-0.038	3.335448	-45.258439	2.413998	-224.10354
19	0.875	0.137	0	0.137	3.537428	17.072049	2.641631	-5.700392
20	0.85	0.137	-0.16	0.273	3.749016	10.434934	2.886984	2.496055
21	0.825	0.141	-0.12	0.24	3.963785	11.324716	3.143445	1.773405
22	0.8	0.144	-0.2	0.304	4.179723	9.820507	3.409242	3.008442
23	0.7	0.164	-0.07	0.213	5.046777	11.859988	4.567078	0.099253
24	0.6	0.171	-0.29	0.345	5.96672	9.605173	5.987225	4.214716
25	0.5	0.2	-0.29	0.345	6.949257	9.605173	7.779309	4.214716
26	0.4	0.229	-0.0025	0.23	7.988528	10.413495	10.102174	-0.741033
27	0	0.23	-	-	-	-	-	-

All of the parameters listed in table 1, though specific to the Lapua Scenar GB528 rifle bullet, are independent of air density and initial speed.

[4] Wikipedia. External Ballistics. http://en.wikipedia.org/wiki/External_ballistics (accessed 3 June 2013).

7.2 Calculating Speed as a Function of Initial Speed and Distance Traveled

For the Lapua Scenar GB528 rifle bullet, Wikipedia's External Ballistics page lists mass (m) as 19.44 g, diameter as 8.59 mm, and initial speed ($v_{initial}$) as 830 m/s.

Assuming standard sea-level atmospheric conditions (air density (ρ) equals 1.225 kg/m^3 and speed of sound (v_s) equals 340.3 m/s)[5] the speed of a bullet that has traveled 300 m (Δx) can be calculated using the equations presented in sections 1 and 5.

First use equation 1 to find initial Mach number ($M_{initial}$):

$$M_{initial} = \frac{v_{initial}}{v_{sound}} = \frac{830\ \text{m/s}}{340.3\ \text{m/s}} = 2.43902 \tag{51}$$

Next, use the equation for the area of a circle to find the cross-sectional area (A) of the projectile:

$$A = \pi r^2 = 3.14159\left(\frac{.00859\ \text{m}}{2}\right)^2 = 5.7953\times10^{-5}\ \text{m}^2 \tag{52}$$

Use equation 2 to find the scaled distance traveled (Δs):

$$\Delta s = \frac{\rho A}{2m}\Delta x = \frac{(1.225\ \text{kg/m}^3)(5.7953\times10^{-5}\ \text{m}^2)}{2(.01944\ \text{kg})}(300\ \text{m}) = .54778 \tag{53}$$

Use equation 45 to find the initial scaled distance ($s_{initial}$):

$$s_{initial} = \frac{1}{\beta_{k=0}}\ln\left(\alpha_{k=0} + \frac{\beta_{k=0}}{M_{initial}}\right) + c_{i=0} = \frac{1}{.414}\ln\left(-.06 + \frac{.414}{2.43902}\right) + 5.2773 = -.059991 \tag{54}$$

Use equation 43 to find the final scaled distance (s_{final}):

$$s_{final} = s_{initial} + \Delta s = -.059991 + .54778 = .487789 \tag{55}$$

Use equation 46 to find the final Mach number (M_{final}):

$$M_{final} = \frac{\beta_{k=1}}{e^{\beta_{k=1}(s_{final} - c_{k=1})} - \alpha_{k=1}} = \frac{.392}{e^{.392(.487789 - 5.555784)} + .05} = 2.09454 \tag{56}$$

Finally, use equation 1 to find the final speed (v_{final}):

$$v_{final} = M_{final} v_{sound} = 2.09454(340.3\ \text{m/s}) = 713\ \text{m/s} \tag{57}$$

[5] National Aeronautics and Space Administration. *U.S. Standard Atmosphere, 1976*; NASA-TM-X-74335; U.S. Government Printing Office: Washington, DC, October 1976.

7.3 Calculating Time of Flight as a Function of Initial Speed and Distance Traveled

Use equation 47 to find $\tau_{initial}$ and τ_{final}:

$$\begin{aligned}\tau_{initial} &= \frac{1}{\beta_{k=0} M_{initial}} - \frac{\alpha_{k=0}}{\beta_{k=0}^2} \ln\left(\alpha_{k=0} + \frac{\beta_{k=0}}{M_{initial}}\right) + d_{k=0} \\ &= \frac{1}{(.414)(2.439)} - \frac{(-.06)}{.414^2} \ln\left(-.06 + \frac{.414}{2.439}\right) + (-.2416) \\ &= -.024768,\end{aligned} \tag{58}$$

$$\begin{aligned}\tau_{final} &= \frac{1}{\beta_{k=1} M_{final}} - \frac{\alpha_{k=1}}{\beta_{k=1}^2} \ln\left(\alpha_{k=1} + \frac{\beta_{k=1}}{M_{initial}}\right) + d_{k=1} \\ &= \frac{1}{(.392)(2.095)} - \frac{(-.05)}{.392^2} \ln\left(-.05 + \frac{.392}{2.095}\right) + (-.3539) \\ &= .217245.\end{aligned} \tag{59}$$

Next, calculate the change in scaled time ($\Delta\tau$):

$$\Delta\tau = \tau_{final} - \tau_{initial} = .217245 - (-.024768) = .242013 \tag{60}$$

Finally, use equation 3 to find time of flight (Δt)

$$\Delta t = \frac{2m\Delta\tau}{\rho A v_s} = \frac{2(.01944\,\text{kg})(.242013)}{(1.225\,\text{kg/m}^3)(5.795\times10^{-5}\,\text{m}^2)(340.3\,\text{m/s})} = .390\,\text{s} \tag{61}$$

---

8. C++ Implementation

8.1 Storing Drag-Table Coefficients: The DRAG Struct

DRAG structs are used to store the parameters associated with a single linear drag profile. Multiple DRAG structs can be used to describe a nonlinear drag profile.

DRAG Code

```
struct DRAG{//<==============A LINEAR DRAG PROFILE (CREATE A SET USING Drags())
    double M;//<-------------------THE MAXIMUM MACH VALUE FOR THE DRAG PROFILE
    double a;//<---------------------------------THE SLOPE OF THE DRAG PROFILE
    double b;//<---------------------------THE Y-INTERCEPT OF THE DRAG PROFILE
    double s;//<--------------THE MINIMUM SCALED DISTANCE FOR THE DRAG PROFILE
    double c;//<-------------------THE SCALED-DISTANCE CONSTANT OF INTEGRATION
    double T;//<------------------THE MINIMUM SCALED TIME FOR THE DRAG PROFILE
    double d;//<-----------------------THE SCALED-TIME CONSTANT OF INTEGRATION
};//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

DRAG Parameters

M **M** specifies the maximum Mach value for the drag profile.

a **a** specifies the slope, α, of the drag profile (see equation 28).

b **b** specifies the y-intercept, β, of the drag profile (see equation 29).

s **s** specifies the minimum scaled distance, s_0, for the drag profile (see equations 33–35).

c **c** specifies *c*, the integration constant from equation 31.

T **T** specifies the minimum scaled time, τ, for the DRAG profile (see equations 39–41).

d **d** specifies *d*, the integration constant from equation 38.

8.2 Creating a Pointer to an Array of DRAG Structs: The Drags() Function

The Drags() function can be used to create a pointer to an array of DRAG structs. The code uses equation 32 to determine $M_{0,k}$; equation 28 to determine α_k; equation 29 to determine β_k; equations 33–35 to determine $s_{0,k}$; equation 31 to determine c_k; equations 39–41 to find τ_k; and equation 38 to determine d_k. If β_k is found to be very close to zero, it is replaced with the user-definable value **b_min**.

Note that the Drags() function uses the "new" command to allocate memory for the array of DRAGs that is pointed to by the return value. Thus, to avoid memory leaks, each use of the Drags() function should be accompanied by a use of the "delete[]" operator.

Drags() Code

```
inline DRAG*Drags(//<==================================CREATES A SET OF DRAGS
    const double*M,//<--------------MACH VALUESS (MUST BE UNIQUE & DECREASING)
    const double*Cd,//<------CORRESPONDING Cd VALUES (FROM F=-.5*rho*A*Cd*v^2)
    int n,//<-----------------------------------------THE NUMBER OF MACH VALUES
    double b_min=1E-8){//<---------------------------------MINIMUM BETA VALUE
  DRAG*D=new DRAG[n-1];
  for(int k=0,j=k-1;k<n-1;++k,++j){
    D[k].M=M[k];
    D[k].a=(Cd[k]-Cd[k+1])/(M[k]-M[k+1]);
    D[k].b=Cd[k]-D[k].a*M[k];/*<-*/if(fabs(D[k].b)<b_min)D[k].b=b_min;
    D[k].s=!k?0:1/D[j].b*log(D[j].a+D[j].b/M[k])+D[j].c;
    D[k].c=D[k].s-log(D[k].a+D[k].b/M[k])/D[k].b;
    D[k].T=!k?0:(1/M[k]-D[j].a/D[j].b*log(D[j].a+D[j].b/M[k]))/D[j].b+D[j].d;
    D[k].d=D[k].T-(1/M[k]-D[k].a/D[k].b*log(D[k].a+D[k].b/M[k]))/D[k].b;}
  return D;//.....................note that D points to newly allocated memory
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

Drags() Parameters

M **M** points to an array of Mach values. The Mach values must be unique and listed in decreasing order.

Cd **Cd** points to an array of drag coefficients. There must be a drag coefficient for each Mach value.

n **n** specifies the number of elements contained in the array that is pointed to by **M**.

b_min **b_min** specifies the smallest value that β_k is allowed to have (see the discussion that follows equation 25).

Drags() Return Value

The Drags() function returns a pointer to an array of DRAG structs.

8.3 Calculating Scaled Distance as a Function of Mach Number: The SofM() Function

The SofM() function uses equation 45 to calculate scaled distance as a function of Mach number.

SofM() Code

```
inline double SofM(//<=============SCALED DISTANCE AS A FUNCTION OF MACH NUMBER
    const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
    int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
    double M){//<-----------------------------------------MACH NUMBER (M=v/c)
  int k;/*<-*/for(k=n-2;k>0&&M>D[k].M;--k);
  return 1/D[k].b*log(D[k].a+D[k].b/M)+D[k].c;//...............s=rho*A/(2*m)*x
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

SofM() Parameters

D **D** points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

n **n** specifies the number of Mach elements used to create **D**. Note that **n** is one less than the size of **D**.

M **M** specifies a Mach number. Use equation 1 to convert from speed to Mach number.

SofM() Return Value

The SofM() function returns a scaled distance. Use equation 2 to convert from scaled distance to measured distance.

8.4 Calculating Mach Number as a Function of Scaled Distance: The MofS() Function

The MofS() function uses equation 46 to calculate Mach number as a function of scaled distance.

MofS() Code

```
inline double MofS(//<============MACH NUMBER AS A FUNCTION OF SCALED DISTANCE
    const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
    int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
    double s){//<----------------------------SCALED-DISTANCE (s=rho*A/(2*m)*x)
  int k;/*<-*/for(k=n-2;k>0&&s<D[k].s;--k);
  return D[k].b/(exp(D[k].b*(s-D[k].c))-D[k].a);//.....................M=v/c
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

MofS() Parameters

D **D** points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

n **n** specifies the number of Mach elements used to create **D**. Note that **n** is one less than the size of **D**.

s **s** specifies a scaled-distance. Use equation 2 to convert from measured distance to scaled distance.

MofS() Return Value

The MofS() function returns a Mach number. Use equation 1 to convert from Mach number to speed.

8.5 Calculating Scaled Time: The TofM() Function

The TofM() function uses equation 47 to calculate scaled time as a function of Mach number.

TofM() Code

```
inline double TofM(//<================SCALED TIME AS A FUNCTION OF MACH NUMBER
    const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
    int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
    double M){//<-----------------------------------------MACH NUMBER (M=v/c)
  int k;/*<-*/for(k=n-2;k>0&&M>D[k].M;--k);//          T=rho*A/(2*m)*c*t ----.
  return(1/M-D[k].a/D[k].b*log(D[k].a+D[k].b/M))/D[k].b+D[k].d;//<-----------'
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

TofM() Parameters

D **D** points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

n **n** specifies the number of Mach elements used to create **D**. Note that **n** is one less than the size of **D**.

M **M** specifies a Mach number. Use equation 1 to convert from speed to Mach number.

TofM() Return Value

The TofM() function returns a scaled time. Use equation 3 to convert from scaled time to measured time.

8.6 Example: Calculating Speed and Time of Flight for the Lapua Scenar GB528 Rifle Bullet

The following example uses information from Wikipedia's External Ballistics page[4] to calculate speed and time of flight as functions of total distance traveled for a Lapua Scenar GB528 rifle bullet. Calculated values are compared to Doppler radar measurements, which were also obtained from Wikipedia's External Ballistics page.

```
#include <cstdio>//.....................................................printf()
#include "y_traj_1d.h"//...............................yTraj1D,<cmath>{fabs()}
int main(){
  double M[]={2.4,2.2,2,1.8,1.6,1.5,1.4,1.3,1.2,1.15,1.1,1.075,1.050,1.025,1,
    .975,.950,.925,.9,.875,.85,.825,.8,.7,.6,.5,.4,0};
  double C[]={.270,.282,.292,.304,.321,.328,.336,.343,.348,.348,.347,.345,.341,
    .334,.306,.236,.177,.154,.142,.137,.137,.141,.144,.164,.171,.2,.229,.23};
  yTraj1D::DRAG*D=yTraj1D::Drags(M,C,28);
  double A=3.14159*pow(8.59/2000,2);//..cross-sectional area of projectile (m^2)
  double m=19.44/1000;//.................................mass of projectile (kg)
  double v0=830;//..............................initial speed of projectile (m/s)
  double rho=1.225;//.........................density of air at sea level (kg/m^3)
  double c=340.3;//...................................local speed of sound (m/s)
  double v_r[]={830,711,604,507,422,349,311,288,267,247,227};
  double t_r[]={0,.3918,.8507,1.3937,2.0435,2.8276,3.748,4.7522,5.8254,7.0095,
    8.2909};
  printf("%13s       | calc.    ref.          |   calc.    ref.    \n","");
  printf("%13s range | speed   speed     %%   |   time     time     %%\n","");
  printf("%13s  (m)  | (m/s) , (m/s) , diff. |   (s)  ,   (s)  , diff.\n","");
  printf("%13s=========================================================\n","");
  for(int i=0;i<11;++i){
    double v=yTraj1D::MofS(D,28,yTraj1D::SofM(D,28,v0/c)+300*i*rho*A/(2*m))*c;
    double t=(yTraj1D::TofM(D,28,v/c)-yTraj1D::TofM(D,28,v0/c))*2*m/(rho*A*c);
    printf("%19.1f |%6.1f ,%6.1f ,%5.1f  |%7.4f ,%7.4f ,%5.1f\n",
           300.*i,v,v_r[i],fabs(v_r[i]-v)/((v_r[i]+v)/2)*100,t,t_r[i],
           i==0?0:fabs(t_r[i]-t)/((t_r[i]+t)/2)*100);}
  delete[] D;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

OUTPUT:

```
                  | calc.    ref.          |   calc.    ref.
          range   | speed    speed     %   |   time     time     %
           (m)    | (m/s) , (m/s) , diff.  |   (s)  ,   (s)  , diff.
         ========================================================
            0.0   | 830.0 , 830.0 ,  0.0   | 0.0000 , 0.0000 ,  0.0
          300.0   | 712.7 , 711.0 ,  0.2   | 0.3901 , 0.3918 ,  0.4
          600.0   | 606.3 , 604.0 ,  0.4   | 0.8467 , 0.8507 ,  0.5
          900.0   | 509.3 , 507.0 ,  0.5   | 1.3867 , 1.3937 ,  0.5
         1200.0   | 423.3 , 422.0 ,  0.3   | 2.0333 , 2.0435 ,  0.5
         1500.0   | 350.2 , 349.0 ,  0.3   | 2.8140 , 2.8276 ,  0.5
         1800.0   | 311.1 , 311.0 ,  0.0   | 3.7319 , 3.7480 ,  0.4
         2100.0   | 288.3 , 288.0 ,  0.1   | 4.7344 , 4.7522 ,  0.4
         2400.0   | 267.7 , 267.0 ,  0.3   | 5.8183 , 5.8254 ,  0.1
         2700.0   | 245.2 , 247.0 ,  0.7   | 6.9894 , 7.0095 ,  0.3
         3000.0   | 223.3 , 227.0 ,  1.7   | 8.2681 , 8.2909 ,  0.3
```

8.7 Example: Calculating Scaled Time of Impact for Fragment Trajectories

The following example simulates the trajectories of fragments produced by a fragmentation weapon.

The code generates 1,000,000 fragments, all with the same starting location. Each fragment is given a randomly chosen velocity. Direction is uniformly distributed across all possible directions. Speed is uniformly distributed with values from Mach 1.0 to Mach 3.0.

A rectangular surface that is 80.0 × 40.0 scaled length units is placed with its center 10.0 scaled length units away from the fragment source. The surface is oriented such that, at its center, it is perpendicular to a ray originating from the fragment source.

Distance to point of impact is calculated using functions from the y3DOps namespace.[6] The MofS() function is used to find each fragment's speed at the impact location. The TofM() function is used to find the scaled time of impact.

```
#include <cstdio>//.....................fclose(),FILE,freopen(),printf(),stdout
#include <cstdlib>//.....................................rand(),RAND_MAX
#include "y_3d_ops.h"//.................................................y3DOps
#include "y_traj_1d.h"//.....................yTraj1D,<cmath>{cos(),sin(),sqrt()}
int main(){
  FILE*f=freopen("impacts.txt","w",stdout);//.........redirect output to a file
  double M[]={3,2.2,1.5,1,.5,.2},C[]={.45,.5,.6,.9,.5,.2};//..............Cd(M)
  yTraj1D::DRAG*D=yTraj1D::Drags(M,C,6);
  double T[]={10,40,20 , 10,-40,20 , 10,40,-20};//............rectangular target
  for(int i=0;i<1000000;++i){
    double M0=2*rand()/RAND_MAX+1;//.............................initial speed
    double phi=2*3.14159265358979*rand()/RAND_MAX;//......random azimuthal angle
    double theta=acos(2.*rand()/RAND_MAX-1);//..........random inclination angle
```

[6] Yager, R. J. *Three-Dimensional Translations, Rotations, and Intersections Using C++*; ARL-TN-557; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2013.

```
    double S[]={0,0,0,sin(theta)*cos(phi),sin(theta)*sin(phi),cos(theta)};
    double t[3];/*<-*/y3DOps::IParameters(t,T,S);//......intersection parameters
    double x[3];/*<-*/y3DOps::Intersect(x,t,S);//..........point of intersection
    double M=yTraj1D::MofS(D,6,yTraj1D::SofM(D,6,M0)
      +sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]));
    double T=yTraj1D::TofM(D,6,M)-yTraj1D::TofM(D,6,M0);//.scaled time of impact
    if(0<t[0]&&0<t[1]&&t[1]<1&&0<t[2]&&t[2]<1)
      printf("%8.3f,%8.3f,%8.3f,%8.3f,%8.3f\n",x[0],x[1],x[2],M,T);}
  fclose(f);
  delete[] D;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

The results are shown in figure 9. Since all time and length units are scaled, the image presented in figure 9 is independent of fragment mass, fragment cross-sectional area, and air density. Note the color mixing that results from selecting random initial speeds.

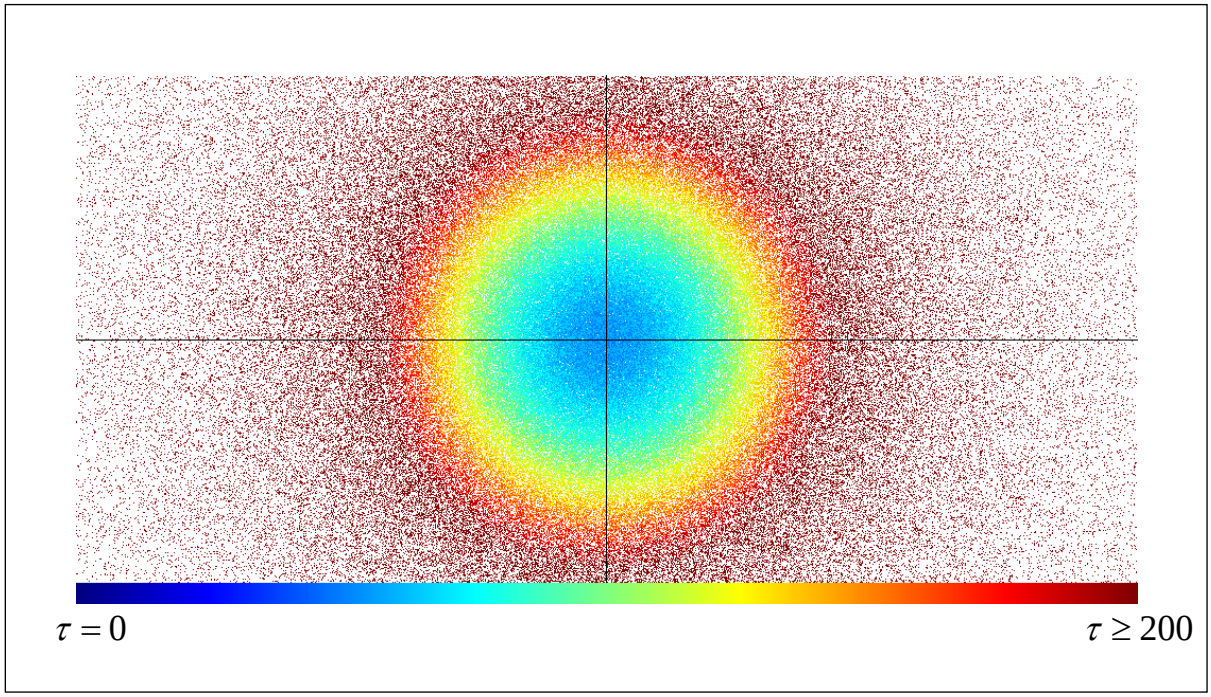


Figure 9. Scaled time of impact, τ , for fragments striking a plate.

9. Code Summary

A summary sheet is provided at the end of this report. It presents the yTraj1D namespace, which contains the struct and four functions that are described in sections 8.1–8.5. Also presented are the examples found in sections 8.6 and 8.7.

yTraj1D Summary

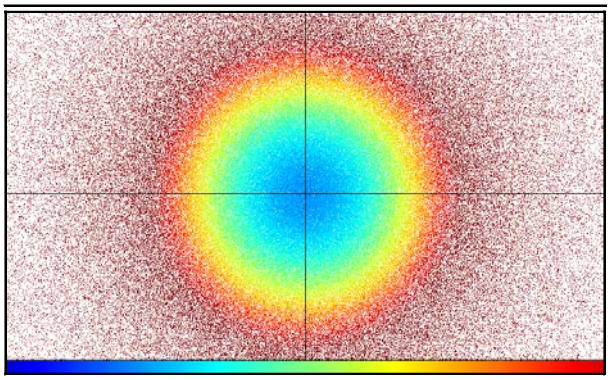
y_traj_1d.h

```
#ifndef Y_TRAJ_1D_GUARD// See Yager, R.; Flanders, B. "A One-Dimensional, Nonit-
#define Y_TRAJ_1D_GUARD//  erative Trajectory Model (with a C++ Implementation)"
#include <cmath>//.....................................exp(),fabs(),log()
namespace yTraj1D{//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ A 1D, NON-ITERATIVE TRAJECTORY MODEL
  struct DRAG{//<=============A LINEAR DRAG PROFILE (CREATE A SET USING Drags())
      double M;//<--------------------THE MAXIMUM MACH VALUE FOR THE DRAG PROFILE
      double a;//<-------------------------------THE SLOPE OF THE DRAG PROFILE
      double b;//<-------------------------------THE Y-INTERCEPT OF THE DRAG PROFILE
      double s;//<--------------THE MINIMUM SCALED DISTANCE FOR THE DRAG PROFILE
      double c;//<--------------------THE SCALED-DISTANCE CONSTANT OF INTEGRATION
      double T;//<--------------------THE MINIMUM SCALED TIME FOR THE DRAG PROFILE
      double d;//<-------------------------THE SCALED-TIME CONSTANT OF INTEGRATION
  };//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
  inline DRAG*Drags(//<=====================================CREATES A SET OF DRAGS
      const double*M,//<--------------MACH VALUESS (MUST BE UNIQUE & DECREASING)
      const double*Cd,//<------CORRESPONDING Cd VALUES (FROM F=-.5*rho*A*Cd*v^2)
      int n,//<-----------------------------------------THE NUMBER OF MACH VALUES
      double b_min=1E-8){//<---------------------------------MINIMUM BETA VALUE
    DRAG*D=new DRAG[n-1];
    for(int k=0,j=k-1;k<n-1;++k,++j){
      D[k].M=M[k];
      D[k].a=(Cd[k]-Cd[k+1])/(M[k]-M[k+1]);
      D[k].b=Cd[k]-D[k].a*M[k];/*<-*/if(fabs(D[k].b)<b_min)D[k].b=b_min;
      D[k].s=!k?0:1/D[j].b*log(D[j].a+D[j].b/M[k])+D[j].c;
      D[k].c=D[k].s-log(D[k].a+D[k].b/M[k])/D[k].b;
      D[k].T=!k?0:(1/M[k]-D[j].a/D[j].b*log(D[j].a+D[j].b/M[k]))/D[j].b+D[j].d;
      D[k].d=D[k].T-(1/M[k]-D[k].a/D[k].b*log(D[k].a+D[k].b/M[k]))/D[k].b;}
    return D;//.....................note that D points to newly allocated memory
  }//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
  inline double SofM(//<=============SCALED DISTANCE AS A FUNCTION OF MACH NUMBER
      const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
      int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
      double M){//<-------------------------------------------MACH NUMBER (M=v/c)
    int k;/*<-*/for(k=n-2;k>0&&M>D[k].M;--k);
    return 1/D[k].b*log(D[k].a+D[k].b/M)+D[k].c;//...............s=rho*A/(2*m)*x
  }//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
  inline double MofS(//<=============MACH NUMBER AS A FUNCTION OF SCALED DISTANCE
      const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
      int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
      double s){//<-----------------------------SCALED-DISTANCE (s=rho*A/(2*m)*x)
    int k;/*<-*/for(k=n-2;k>0&&s<D[k].s;--k);
    return D[k].b/(exp(D[k].b*(s-D[k].c))-D[k].a);//.......................M=v/c
  }//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
  inline double TofM(//<=================SCALED TIME AS A FUNCTION OF MACH NUMBER
      const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
      int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
      double M){//<-------------------------------------------MACH NUMBER (M=v/c)
    int k;/*<-*/for(k=n-2;k>0&&M>D[k].M;--k);//          T=rho*A/(2*m)*c*t ----.
    return(1/M-D[k].a/D[k].b*log(D[k].a+D[k].b/M))/D[k].b+D[k].d;//<-----------'
  }//~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

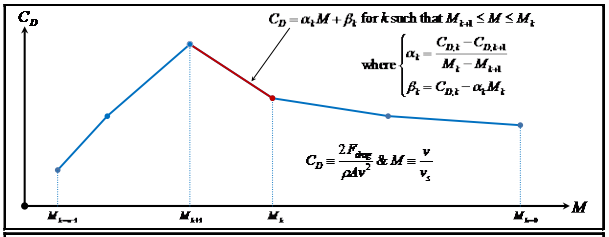
"Impacts" Example

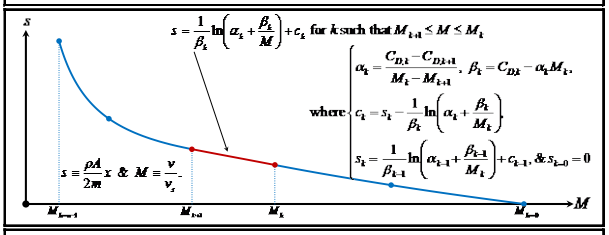
```
#include <cstdio>//.....................fclose(),FILE,freopen(),printf(),stdout
#include <cstdlib>//.....................................rand(),RAND_MAX
#include "y_3d_ops.h"//.....................................y3DOps
#include "y_traj_1d.h"//.....................yTraj1D,<cmath>{cos(),sin(),sqrt()}
int main(){
  FILE*f=freopen("impacts.txt","w",stdout);//..........redirect output to a file
  double M[]={3,2.2,1.5,1,.5,.2},C[]={.45,.5,.6,.9,.5,.2};//...............Cd(M)
  yTraj1D::DRAG*D=yTraj1D::Drags(M,C,6);
  double T[]={10,40,20 , 10,-40,20 , 10,40,-20};//.............rectangular target
  for(int i=0;i<1000000;++i){
    double M0=2*rand()/RAND_MAX+1;//.............................initial speed
    double phi=2*3.14159265358979*rand()/RAND_MAX;//.....random azimuthal angle
    double theta=acos(2.*rand()/RAND_MAX-1);//..........random inclination angle
    double S[]={0,0,0,sin(theta)*cos(phi),sin(theta)*sin(phi),cos(theta)};
    double t[3];/*<-*/y3DOps::IParameters(t,T,S);//.....intersection parameters
    double x[3];/*<-*/y3DOps::Intersect(x,t,S);//..........point of intersection
    double M=yTraj1D::MofS(D,6,yTraj1D::SofM(D,6,M0)
      +sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]));
    double T=yTraj1D::TofM(D,6,M)-yTraj1D::TofM(D,6,M0);//.scaled time of impact
    if(0<t[0]&&0<t[1]&&t[1]<1&&0<t[2]&&t[2]<1)
      printf("%8.3f,%8.3f,%8.3f,%8.3f,%8.3f\n",x[0],x[1],x[2],M,T);}
  fclose(f);
  delete[] D;
}//~~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
```

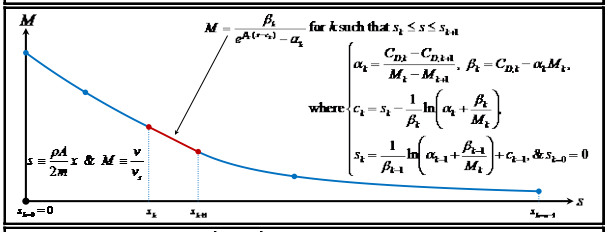
Graph of Output from "Impacts" Example

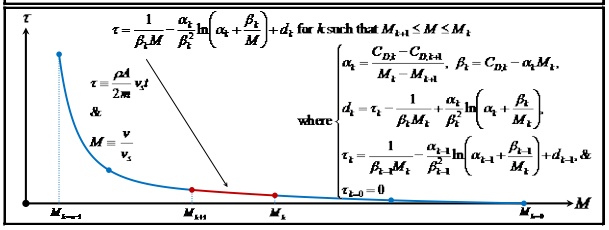


Equations









"Validating" Example

```
#include <cstdio>//.....................................................printf()
#include "y_traj_1d.h"//.................................yTraj1D,<cmath>{fabs()}
int main(){
  double M[]={2.4,2.2,2,1.8,1.6,1.5,1.4,1.3,1.2,1.15,1.1,1.075,1.050,1.025,1,
    .975,.950,.925,.9,.875,.85,.825,.8,.7,.6,.5,.4,0};
  double C[]={.270,.282,.292,.304,.321,.328,.336,.343,.348,.348,.347,.345,.341,
    .334,.306,.236,.177,.154,.142,.137,.137,.141,.144,.164,.171,.2,.229,.23};
  yTraj1D::DRAG*D=yTraj1D::Drags(M,C,28);
  double A=3.14159*pow(8.59/2000,2);//..cross-sectional area of projectile (m^2)
  double m=19.44/1000;//.................................mass of projectile (kg)
  double v0=830;//..............................initial speed of projectile (m/s)
  double rho=1.225;//.......................density of air at sea level (kg/m^3)
  double c=340.3;//.................................local speed of sound (m/s)
  double v_r[]={830,711,604,507,422,349,311,288,267,247,227};
  double t_r[]={0,.3918,.8507,1.3937,2.0435,2.8276,3.748,4.7522,5.8254,7.0095,
    8.2909};
  printf("%13s       | calc.    ref.         |  calc.    ref.    \n","");
  printf("%13s range | speed    speed     %%  |  time      time     %%\n","");
  printf("%13s  (m)  | (m/s) , (m/s) , diff. |  (s)  ,   (s)  , diff.\n","");
  printf("%13s=========================================================\n","");
  for(int i=0;i<11;++i){
    double v=yTraj1D::MofS(D,28,yTraj1D::SofM(D,28,v0/c)+300*i*rho*A/(2*m))*c;
    double t=(yTraj1D::TofM(D,28,v/c)-yTraj1D::TofM(D,28,v0/c))*2*m/(rho*A*c);
    printf("%19.1f |%6.1f ,%6.1f ,%5.1f  |%7.4f ,%7.4f ,%5.1f\n",
           300.*i,v,v_r[i],fabs(v_r[i]-v)/((v_r[i]+v)/2)*100,t,t_r[i],
           i==0?0:fabs(t_r[i]-t)/((t_r[i]+t)/2)*100);}
  delete[] D;
}//~~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~~
```

Output from "Validating" Example

range (m)	calc. speed (m/s)	ref. speed (m/s)	% diff.	calc. time (s)	ref. time (s)	% diff.
0.0	830.0	830.0	0.0	0.0000	0.0000	0.0
300.0	712.8	711.0	0.2	0.3901	0.3918	0.4
600.0	606.1	604.0	0.3	0.8466	0.8507	0.5
900.0	509.3	507.0	0.5	1.3867	1.3937	0.5
1200.0	423.2	422.0	0.3	2.0333	2.0435	0.5
1500.0	350.2	349.0	0.3	2.8140	2.8276	0.5
1800.0	311.2	311.0	0.0	3.7319	3.7480	0.4
2100.0	288.2	288.0	0.1	4.7344	4.7522	0.4
2400.0	266.6	267.0	0.1	5.8163	5.8254	0.2
2700.0	245.1	247.0	0.8	6.9894	7.0095	0.3
3000.0	224.0	227.0	1.3	8.2697	8.2909	0.3

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL WML A R YAGER